Tarkempi materiaalinhallinta pilvipohjaisella dynaamisella datavisualisoinnilla

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

Riihimäki, Tieto- ja viestintätekniikka

Kevät 2018

Joni Järvenpää

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tieto- ja viestintätekniikka
Riihimäki

Tekijä Joni Järvenpää **Vuosi** 2018

Työn nimi Tarkempi materiaalinhallinta pilvipohjaisella dynaamisella datavisualisoinnilla

Työn ohjaaja/t Toni Laitinen, Jari Mustajärvi

TIIVISTELMÄ

Tarkalla materiaalinhallinnalla pystytään luomaan suoraa taloudellista hyötyä organisaatioille vähentämällä hävikkiä ja parantamalla toimitusvarmuutta. Tietotekniikka on jatkuvasti kehittyvä ala, jossa yritysten on tärkeää pystyä tarjoamaan aina viimeisintä käytössä olevaa teknologiaa. Tämä luo tarpeen jatkuvalle varastosaldojen tarkastelulle, jotta voidaan minimoida komponenttien vanhentuminen ja saada tieto komponenttitarpeista mahdollisimman ajoissa.

Tämän opinnäytetyön tarkoituksena on luoda asiakasyritykselle tarkempi materiaalinhallintaohjelmisto ASP.NET web-ohjelmistokehystä käyttäen. Työssä käydään myös läpi MVC- arkkitehtuuria ja sen tukemiseen käytettyjä ohjelmistokieliä, kuten Razor-syntaksia.

Käytännön osuus toteutettiin luomalla sähköinen keräyslista ohjelmistotuote Elkome Installaatiot Oy:lle, joka toimii yrityksen lähiverkon IIS-palvelimelta. Kyseistä ohjelmistoa yrityksen työntekijät pystyvät käyttämään niin yrityksen tietokoneilta kuin tableteilta.

Avainsanat Pilvipalvelut, hallintajärjestelmät, käyttöliittymät, LEAN-ajattelu

Sivut 22

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Information and communication technology
Riihimäki

Author Joni Järvenpää **Year** 2018

Subject Improved material management via cloud-based data projection

Supervisors Toni Laitinen, Jari Mustajärvi

ABSTRACT

An organization can gain direct financial benefit by improving their material management. This will decrease material loss and improve delivery reliability. ICT is continuously evolving, which means the organizations working in the field must be prepared to offer the latest technology available in the market. This creates a constant necessity for storage monitoring, so that the organization can minimize electrical components outdating and to receive information of component demand as quickly as possible.

The goal of this thesis is to improve the client organizations material management using a ASP.NET web-framework. This thesis will also include basic functionality of the MVC software architecture and the programming languages used to support the software, such as the Razor-syntax.

The practical part of this thesis was made by creating a “Software Gathering notes” product for Elkome Installaatiot Oy that is hosted in the local networks IIS server. The employees of the organization can therefore use the software from company computers and tablets.

Keywords Cloud services, management software, interfaces, LEAN-thinking

Pages 22

SISÄLLYS

1 SANASTO

MVC	Model-View-Controller- arkkitehtuuri (Malli, näkymä, käsittelijä)
Käsittelijä	MVC rakenteen käsittelijä osa (Controller), joka hoitaa websovelluksen toiminnallisen osuuden.
Malli	MVC rakenteen model-osio, johon keskitetään tietylle tie tojoukkiolle ominainen logiikka.
Näkymä	MVC rakenteen Front-end. Käyttäjälle näkyvä websivusto, jossa hyödynnetään Razor-syntaksia.
Back-end	Sovelluksen taustalla toimiva logiikka, jota käyttäjä ei konk reettisesti näe.
Front-end	Sovelluksen visuaalinen osa, jonka käyttäjä voi nähdä. Toiminnallisuudet on useimmiten keskitetty back-endiin.
Autentikointi	Käyttäjän varmentaminen, sisäänkirjautuminen
cshtml	Razor-syntaksia hyödyntävän websivuston tunniste. Esim. MVC-rakenteen *”näkymä”* -osio

2 TYÖN TAVOITTEET

Asiakasyrityksen projektiluonteisessa tuotevalmistuksessa projektin kulku on pyritty tekemään mahdollisimman selkeästi hyödyntämällä LEAN-ajattelua, jolla pyritään poistamaan prosessista tuottamattomia toimintoja. Tämän tarkoituksena on tehostaa toimintaa ja nopeuttaa prosessin kulkua. Projektiin tarvittavat tuotteet kartoitetaan ja kirjataan myyntitilaukseen. Myyntitilauksen tuoteriveistä muodostetaan paperinen keräyslähete, jonka avulla kerätään tuotteita päävarastosta projektihyllyyn. Projektihyllystä asentaja hakee tuotteita tarpeen mukaan ja asentaa niitä valmistuvaan tuotteeseen.

Elektroniikkakomponentit tyypillisesti tilataan toimittajalta suoraan projektille. Tällä tavoin yritys pystyy tarjoamaan tuoreimmat komponentit näin nopeasti kehittyvällä alalla. Tuotteet jotka eivät vanhene yhtä nopeasti varastoidaan päävarastoon. Paperisessa keräyslistassa ei huomioida sitä, onko tuote päävarastossa vai onko tuote tilattu projektille. Tämän lisäksi paperinen keräyslista ei näytä varaston saldoja kyseiselle tuotteelle, joten projektin tuotteiden kokonaiskuva jää epäselväksi. Paperiselle keräyslistalle merkitään kuulakärkikynällä varastosta projektihyllyyn viety tuote. Keräyslähetteitä voi olla myös useampia, mikä voi johtaa saman tuotteen kahteen kertaan keräilyn. Ongelmana on joidenkin projektien laajuus. Projektit voivat toisinaan olla yli tuhat riviä pitkiä, mikä aiheuttaa miltei 60-sivuisia keräyslähetteitä. Vaikka prosessi itsessään onkin toimiva, on käytännönosuus paperisella keräyslistalla ongelmallinen.

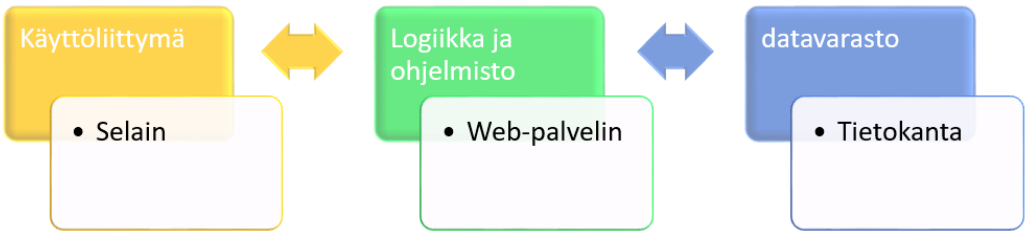
Prosessin nopeuttamiseksi luotiin paperisen keräyslistan tilalle sähköinen keräyslista. Sähköisen keräyslistan toimintatavoitteena on prosessin nopeuttaminen visualisoimalla toiminnanohjausjärjestelmästä dataa työntekijöille, jolla pyritään vähentämään toimihenkilöiltä tiedon kyselyä ja avartamaan projektin kokonaiskulkua työntekijälle. Projektien keräyshallinnoinnilla pystytään vaikuttamaan myös komponenttien keräyksen priorisointiin avaamalla kiireellisemmät projektit varastotyöntekijöille ensimmäisenä. Esteettömän pääsyn vuoksi sähköinen keräyslista toimii verkkoselaimessa sijoitettuna lähiverkkoon, joka helpottaa työntekijöiden pääsyä applikaatioon.

Varastotyöntekijä merkitsee keräyslistaan kerätyn tuotteen. Kerätty tuote näkyy jokaiselle projektiin kuuluvalle henkilölle keräyshetkestä eteenpäin. Tällä pyritään vähentämään sekaannusta kerättyjen ja keräämättömien tuotteiden välillä. Sähköinen keräyslista käyttää tietoa saapuneista tai saapuvista tuotteista toiminnanohjausjärjestelmän tietokannasta. Tällä tavoin projektia kasaava asentaja voi tarkkailla kriittisten tuotteiden saapumista itse keräyslistalta, eikä työntekijän tarvitse keskeyttää työntekoa ottaakseen yhteyttä projektipäällikköön. Kyseinen toiminto tukee yrityksen LEAN-ajattelutapaa. Sähköisen keräyslistan muihin tavoitteisiin kuuluu myös helpompi ja tarkempi projektimonitorointi. Projektipäällikkö pystyy seuraamaan projektinsa kulkua päävalikosta, jossa jokainen projekti on kirjattuna selkeään listaan. Listan nimikkeen alapuolella on etenemispalkki, josta käy ilmi projektin kerätyt tuotteet, kerättävissä olevat tuotteet, tilatut tuotteet ja tilaamattomat tuotteet. Tällä tavoin esimerkiksi tilaamattomiin tuotteisiin pystytään puuttumaan välittömästi. Kokonaisuutena sähköisellä keräyslistalla pyritään korvaamaan entinen paperinen keräyslista, tehostamaan projektin etenemistä ja luomaan parempi projektimonitorointi vastuuhenkilöille.

3 TIETOKANTAA HYÖDYNTÄVÄ WEBSOVELLUS

Sovelluksia on monenlaisia ja ohjelmistokehitys on syytä suunnitella usein käyttötapauskohtaisesti. Eri käyttötarkoitukseen voi lokaalisti asennettava sovellus olla oikea valinta, mutta kun käyttäjäkuntana on useampi henkilö, on syytä harkita verkon yli käytettävää sovellusta. Websovelluksissa hyödyt korostuvat erityisesti päivitettävyys ja itse ohjelmiston asennusvaiheen puuttuminen. Websovelluksen käyttäjät ovat yhteydessä yhteen osoitteeseen, johon ohjelmisto on keskitetty. Tällä tavoin jokaisella sovelluksen käyttäjällä on sama sovellusversio, joka puolestaan alentaa esimerkiksi mahdollisen tietokantavirheiden tapahtumista.

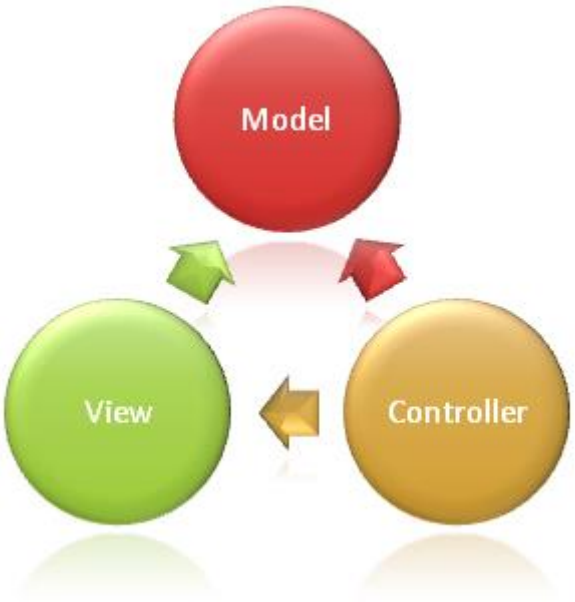
Perusrakenne websovelluksessa on kolmiosainen. Käyttöliittymä on useimmissa tapauksissa rakennettu joko HTML tai PHP pohjaisesti ja ohjelman käyttö tapahtuu selaimen kautta. Käyttöliittymän koodi ja määritykset sijaitsevat puolestaan web-palvelimessa johon käyttäjä ottaa selaimellaan yhteyden. Windows pohjaisessa ympäristössä web-palvelimena toimii useimmiten Microsoftin Internet Information Service- palvelin, eli IIS. Linux pohjaisessa ympäristössä palvelua ylläpitää puolestaan Apache2- palvelin. Web-palvelin kommunikoi ja luo yhteyden tietokantaan, joka monelle websovellukselle on tyypillinen.



Kuva 1. Webohjelmiston komponentit

3.1 **ASP.NET**

Microsoft on kehittänyt web-ohjelmistokehyksensä websovelluksille tai palveluille, mikä mahdollistaa dynaamisten websivujen luomisen. Opinnäytetyössä käytetty ASP.NET-arkkitehtuuri on nimeltään MVC, eli Model-View-Controller- arkkitehtuuri, jossa ohjelmiston rakenne on eriytetty eri komponentteihin kehittämisen helppouden parantamiseksi. Näihin komponentteihin kuuluu malli (model), näkymä (view) ja käsittelijä (controller). (Microsoft 2018a)



Kuva 2. MVC rakenne

Ylläolevassa kuvassa (Kuva 2) on kuvattuna MVC-arkkitehtuurin rakenne, jossa käsittelijä (controller) tekee työtä mallin (model) kanssa ohjatakseen ja suorittaakseen käyttäjän tekemiä toimintoja tai tekemään kantahakuja tai muutoksia tietokantaan.

3.1.1 **Käsittelijä**

Käsittelijä päättää minkä näkymän tuo itse käyttäjän nähtäväksi oman ohjelmoidun logiikkansa perusteella. Käsittelijäkomponenttia MVC-arkkitehtuurissa voidaan ajatella loogisena backend-toimijana, joka ohjaa ja suorittaa ohjelmiston toiminnallisia osuuksia.

3.1.2 **Malli**

Käsittelijä käyttää hyödykseen malleja (model) tehdäkseen esimerkiksi valmiita tietojoukkoja, jotka käsittelijä vie näkymälle (view). Useimmiten eri tyyppisten tietojoukkioiden logiikka sijaitsee itse mallissa, esimerkiksi tietokannasta haettu data viedään mallille ja mallin sisällä tapahtuu tarvittava laskettava logiikka. Laskettu tieto viedään tämän jälkeen näkymälle. Malli on pitkälti samanlainen, kuin normaalissa C# tai Java rakenteessa esiintyvä luokka.

3.1.3 **Näkymä**

Itse näkymäosuuden vastuualue on visualisoida tuotu data käyttäjälle helppolukuiseen muotoon. Näkymäosuus voi käyttää hyväkseen perinteisiä webtekniikoissa käytettyjä kieliä, esimerkiksi HTML, CSS ja Javascript kieliä. ASP.NET MVC- arkkitehtuurissa käytetty tiedostomuoto näkymäkomponentissa on cshtml. Kyseinen tiedostomuoto kielii tiedoston sisältävän MVC:lle tunnuksenomaista *Razor-syntaksia.*

3.1.4 **Razor-syntaksi**

ASP.NET MVC- arkkitehtuurissa ohjelmoija voi käyttää myös hyödykseen Razor-syntaksia, jonka avulla voidaan hyödyntää ja esittää käsittelijältä saatu data muuttujanomaisesti websivustolla. Razor-syntaksin ansiosta ohjelmoija voi käyttää C#-kieltä näkymäkomponentissa purkaakseen käsittelijältä tuodun datan verkkosivustolle. Razor-syntaksia voidaan myös käyttää esimerkiksi luomaan C# kielen ominaisia toiminnallisuuksia, kuten silmukoita ja ehtoja. Nämä ominaisuudet mahdollistavat dynaamisen websivuston luomisen, jossa käyttäjä voi luoda ehtoja datan esilletuomiseen. Cshtml tiedosto ei estä ohjelmoijaa käyttämästä mitään html websivustolle mahdollista kieltä, kuten CSS:ssää, javascriptiä tai jQuerytä. Razor-syntaksin alkamisen tunnistaa html-koodin seassa sijaitsevasta @-merkistä. Alla olevassa esimerkissä luodaan listarakenne cshtml sivustolle hyödyntäen Razor-syntaksia. Sivuston käsittelijä on lähettänyt näkymäosiolle mallin. Malli sisältää listan objekteja, jotka ovat erään moottoripyöräreittisovelluksen tietokannan dataa.

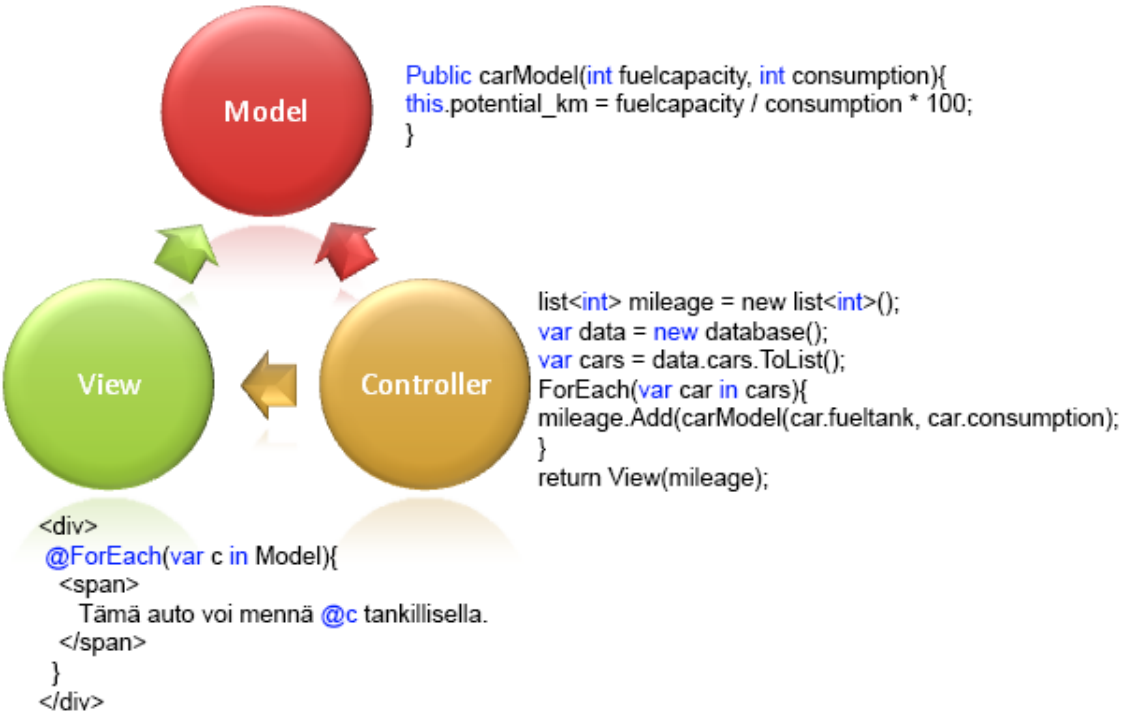
```
<div id="scroll-container">
     <div class="wrap-container" id="wrap-scroll">
          <ul id="ul-scroll">
              @foreach (var route in Model)
               {
                   if (route.type == 1)
                   {
                      <li>
                       <span class="item">
                          <a href="@route.roadurl" name="@route.id" target="route-
                          Map" onclick="changeRoadInfo('@route.roadname',
                          '@route.city', '@route.option', @route.id, @route.rate,
                          @route.userratedroute, @route.userownsroute)">
                          @route.city / @route.roadname
                          <code class="score">@route.rate</code>/ @route.road-
                          length min
                          </a>
                       </span>
                      </li>
                   }
                   else if (route.type == 2)
                   {
                    <li>
                       <span class="item">
                          <a href="@route.roadurl" target="routeMap" on-
                          click="changeRoadInfo('@route.roadname', '@route.city',
                          '@route.option', @route.id, @route.rate, @route.userrat-
                          edroute, @route.userownsroute)">@route.city /
                          @route.roadname
                          <code class="score">@route.rate</code>/ Pysähtymispaikka
                          </a>
                       </span>
                     </li>
                   }
                  }
              </ul>
         </div>
</div>
```

Ylläolevassa esimerkissä luodaan suuresta tietojoukkiosta listan elementtejä hyödyntäen *ForEach* ja *if* toimintoa. Käsittelijän lähettämää mallin listaa puretaan *ForEach* toiminnolla pienempiin elementteihin, jonka muuttujia tuodaan esiin @ etuliitteellä. Alla näet esimerkkikoodin tekemän konkreettisen tuotoksen.

Kuva 3. Reittisovelluksen listaus

3.1.5 Yhteenveto

Kokonaisuudessaan MVC-arkkitehtuuri luo loistavan kehitysympäristön websovelluksille. Alla olevassa kuvassa on kuvattu havainnollistamalla eri MVC-komponenttien toiminalliset osuudet lyhyin koodiesimerkein.



Kuva 4. MVC rakenteen havainnolistaminen koodiesimerkein.

3.2 Microsoft SQL Server

Ohjelmistokokonaisuutta luodessa Windows pohjalle on luonnollista rakentaa alusta yhtenäiseksi. Tietokantaosuus Microsoft pohjaisessa websovelluksessa useimmiten toteutetaankin Microsoft SQL tietokannalla.

3.2.1 Yleistä SQL tietokannoista

Tietokantoja käytetään varastoimaan erilaisia tietoattribuutteja, jotka ovat ohjelman toiminnallisuuden kannalta olennaisia. Tietokannoista voidaan helposti hakea tietojoukkioita loppukäyttäjälle eri toimintatarkoituksissa. Tietokannat muistuttavat rakenteeltaan Excel-taulukkoa, jossa ylimmällä rivillä on attribuutteja, ja niiden alapuolella tietojoukkio rivejä.

3.2.2 Structured Query Language

Tiedon haku, muuttaminen tai lisääminen tapahtuu SQL tietokannoissa SQL lauseella (Structured Query Language).

SQL lauseet ovat useimmissa tilanteissa melko selkokielisiä ja helppokäyttöisiä. Esimerkki:

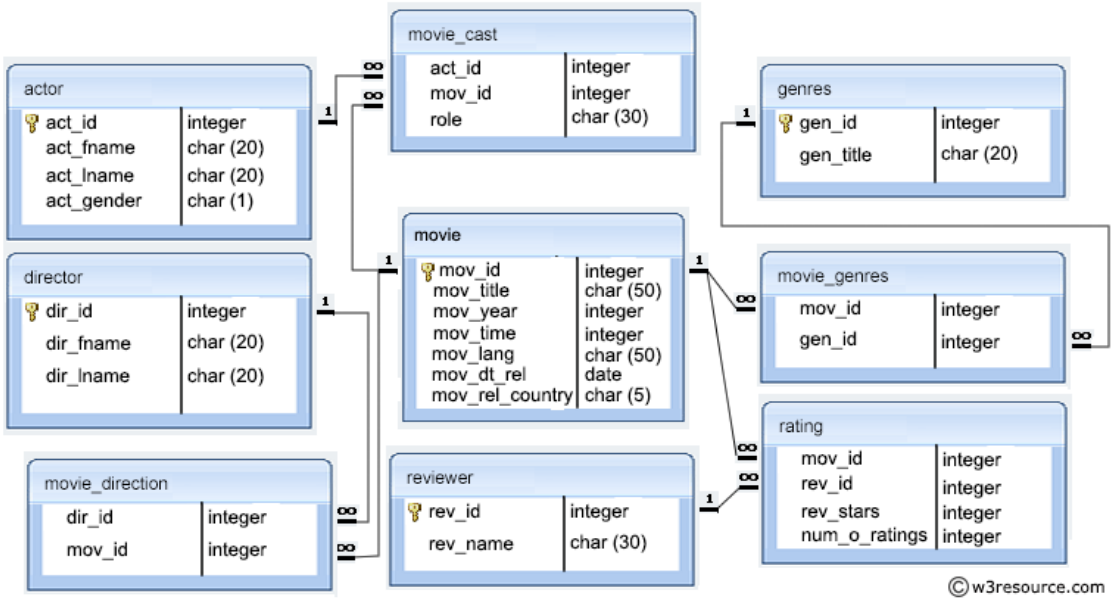
```
SELECT FirstName, LastName, Age  FROM Persons WHERE Age =
20 ORDER BY FirstName DESC;
```

Ylläolevassa esimerkissä haetaan ’*Persons*’ nimisestä tietokantataulukosta henkilöiden etunimi, sukunimi ja ikä tilanteessa, jossa henkilön ikä on 20. Tämän jälkeen taulukko listataan laskevassa järjestyksessä etunimen mukaan. Kyseinen SQL lause vaatii taulukon, jossa taulukon attribuutteihin kuuluu *FirstName, LastName* ja *Age*. Kyseinen haku voisi mahdollisesti tuottaa seuraavanlaisen tietonäkymän käyttäjälle:

FirstName	LastName	Age
Erkki	Esimerkki	20
Teppo	Testaaja	20
Matti	Meikäläinen	20

3.2.3 **Relaatiotietokanta**

Useimmissa tilanteissa tietokannat ovat niin kutsuttuja relaatiotietokantoja ja vaihtoehtoisesti harvemmissa tilanteissa, kanta perustuu oliomalliin. Relaatiotietokanta on taulukkorakennemainen tietokanta, jossa käyttäjä määrittelee tietokannan attribuutit ja riveille tallennetaan attribuuttia vastaava arvo. Jokaisessa taulussa voidaan määritellä yksi arvo taulun pääavaimeksi. Kyseisellä avaimella voimme luoda suhteita eri taulujen välille. Nämä *relaatiot* toimivat osana tietokannan logiikkaa ja selkeyttävät tiedon varastoinnin rakennetta.
Esimerkki:



Kuva 5. Esimerkki tietokantarakenne

Ylläolevassa kuvassa [Kuva 3] on kuvattu *Movie* niminen tietokanta, jossa taulukot ovat kuvassa näkyvät laatikot. Jokaisen laatikon otsikko vastaa taulun nimeä. Kuvasta voimme havainnoida, että tietoa on osioitu eri taulukoihin niiden ominaisuuksien perusteella. Esimerkiksi *actor* taulussa attribuutteina on näyttelijän id, etunimi, sukunimi ja sukupuoli, kun taas *movie* taulussa on attribuutteina id, nimi, vuosi, kieli ja muuta elokuvalle ominaista tietoa.

3.3 **LINQ**

Riippuen websovelluksen arkkitehtuurista ja toteutustavasta, voi tietokannan hakukieli vaihdella. LINQ (Language Integrated Query) on Microsoftin .NET- komponenttikirjaston osa, jolla voidaan suorittaa erilaisia kyselyitä eri käyttökohteisiin. LINQ:illä voidaan esimerkiksi suodattaa ohjelmassa esiintyviä listoja, valita taulukossa olevia attribuutteja ehtolauseiden avulla, tai suorittaa kyselyitä suoraan relaatiotietokannan tauluihin. ASP.NET- ohjelmiston koodissa voit useissa tilanteissa törmätä LINQ-kielen seuraaviin lauseisiin: Select, Where, Contains, First ja ToList. Where- ja Select- toimintoihin on tunnuksenomaista, että niissä alustetaan väliaikainen hallinnointimuuttuja. Tämä muistuttaa pitkälti C#-kielessä olevan ForEach- toiminnon elementin nimeämistä. (Anders Hejlsberg 2007)

Alla olevissa esimerkeissä käydään läpi kuvitteellista tietokantaa, joka sisältää useita eri ravintoloita.
ToList-käskyllä voidaan kääntää tietojoukkio listaksi, joka helpottaa joukkion hallinnointia, sillä listoja voidaan hyödyntää esimerkiksi C# kielessä tutulla *ForEach* käskyllä.
Esimerkiksi:

```
var restaurantList = dataBaseModel.Restaurants.ToList();
```

Kyseisessä esimerkissä käännetään tietokantamallin sisäinen tietojoukkio nimeltä *restaurants* listaksi ja asetetaan lista variaabelin *restaurantList* arvoksi.

Where-käskyllä voidaan valita suurestakin listasta pienempi osio ehtojen avulla, joka puolestaan tukee ohjelmaoptimointia ja parantaa latausaikoja.
Esimerkiksi:

```
var greatRestaurants = restaurantList.Where(
r => r.restaurantRating == 5
).ToList();
```

Kyseisellä lauseella voidaan valita listasta nimeltä *restaurantList* vain ne ravintolat, joissa ravintolan attribuutti *restaurantRating* on 5. Kyselyn jälkeen hakutulos käännetään listaksi ja osoitetaan variaabeli *greatRestaurants* arvoksi.

Select- lauseella voimme valita objektilistasta tietyn sarakkeen arvon ja asettaa sen variaabelin arvoksi.
Esimerkiksi:

```
var restaurantNames = greatRestaurants.Select(
r => r.restaurantName
).ToList();
```

Ylläolevassa esimerkissä valitsemme *greatRestaurants* listasta ainoastaan ravintoloiden nimet ja käännämme tietojoukkion listaksi.

Kyseistä listaa voimme tämän jälkeen suodattaa vielä pienemmäksi esimerkiksi **Contains**-lauseen kautta. **Contains-**lause vertaa sitä, että sisältääkö käsiteltävä tietojoukko ohjelman logiikan mukaista arvoa.
Esimerkiksi:

```
var steakRestaurants = restaurantNames.Where(
r => r.Contains(”Steak”)
).ToList();
```

Kyseinen esimerkki suodattaa listaa, jossa on ravintoloiden nimiä ehdolla, jossa nimen pitää sisältää sana ”Steak”. Tämän jälkeen ehdon läpäisseet nimet käännetään listaksi ja asetetaan muuttujaan. Tällöin *steakRestaurants* lista sisältää ainoastaan ravintoloiden nimiä, joissa esiintyy sana ”Steak”.

Ylläolevaan esimerkkiin viitaten, voisimme kuvitella konsoliapplikaatio ympäristössä haluavamme listan ensimmäisen ravintolan nimen. Tämä tapahtuu helposti käyttämällä toimintoa **First.**

```
Console.WriteLine(steakRestaurants.First());
```

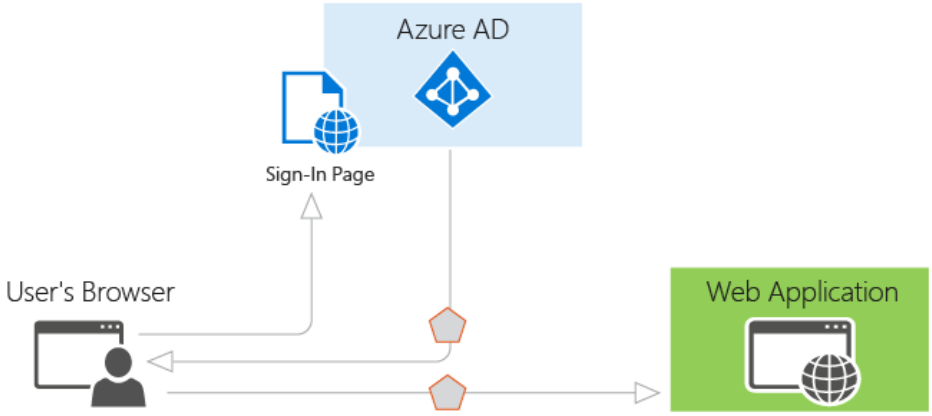
Kyseinen esimerkki kirjoittaa konsoliin *steakRestaurants* listan ensimmäisen sarakkeen. LINQ-kyselykieli on perso ’kasaantumiselle’, sillä kielellä voidaan suorittaa monia kyselyitä ja ehtoja peräkkäin. Esimerkiksi kaikki ylläolevat esimerkit voitaisiin tiivistää yhteen lauseeseen, joka menee seuraavasti:

```
string aNiceSteakRestaurant = dataBaseModel.Restau-
rants.ToList().Where(a => a.restaurantRating = 5 && a.res-
taurantName.Contains(”Steak”)).First().Select(b => b.restau-
rantName;
```

Kyseinen haku tuottaa *string* muuttujaan ensimmäisen ravintolan nimen, jonka *restaurantRating* on 5 ja jonka nimessä esiintyy sana ”Steak”.

3.4 AUTENTIKOINTI

Websovelluksen tarvittaessa kirjautumismetodeja on syytä pohtia eri vaihtoehtoja. Käyttäjän varmentamisen, eli autentikoinnin toteutustapaan vaikuttaa moni tekijä ja näistä yleisimpiä ovat tietoturva ja käyttötarkoitus. Mikäli sovellus sijaitsee tietoturvallisessa ympäristössä, kuten organisaation sisäverkossa, on vähemmän turvallinen autentikointimenetelmä täysin riittävä valinta. Sisäverkossa sijaitsevaan websovellukseen ei pääse vaikuttavaan ulkoiset turvariskit, kuten salasanojen kalastelut tai *bruteforce* hyökkäykset. Myös käyttötarkoitus on painava asia autentikointimenetelmän valinnassa. Esimerkiksi sovelluksessa, jossa halutaan käyttäjän omaavan eri attribuutteja ja oikeuksia voi tietokantapohjainen *token*-autentikointi palvella tarkoitustaan hyvin. Jos käyttäjä halutaan ainoastaan kirjata sisään tietoturvallisesti, jotta käyttäjä pääsisi käsiksi sivuston materiaaliin voidaan miettiä autentikointimenetelmäksi esimerkiksi *Microsft Azuren* tarjoamaa autentikointi vaihtoehtoa, jossa käyttäjä voi kirjautua Office 365, Facebookin tai Googlen tunnuksilla palveluun. (Microsoft 2018b, Microsoft 2018c)



Kuva 6. Azure AD authentication

3.4.1 ASP.NET Forms Authentication

Kyseisessä opinnäytetyössä websovellus sijaitsee organisaation lähiverkossa, jonka vuoksi korkean tietoturvan omaava autentikointimetodi ei ole välttämätön. Organisaation tietokanta, toiminnanohjausjärjestelmä ja IIS-palvelin sijaitsevat kaikki organisaation lähiverkossa, johon ei pääse käsiksi julkisista IP-osoitteista. Microsoftin ASP.NET rakenteeseen kuuluu oma sisäänrakennettu autentikointimenetelmä nimeltään ”.NET Forms Authentication”. Kyseinen autentikointimenetelmä toimii hyvin yksinkertaisissa sovelluksissa, jossa käyttäjän tarvitsee kirjata itsensä sisään päästääkseen käsiksi tiettyyn sisältöön tai toiminnallisuuteen. Forms Authenticationin perusidea on yksinkertainen. Ohjelmoija luo metodin, joka antaa käyttäjälle autentikointi *poletin (eng. Token).* Kyseinen poletti luo käyttäjän sessiolle tilan**:**

```
User.Identiy.IsAuthenticated = true;
```

Tätä autentikointitietoa voidaan käyttää MVC- arkkitehtuurissa esimerkiksi näkymän puolella hyödyntäen Razor-syntaksia vastaavasti:

```
    @if (!User.Identity.IsAuthenticated)
    {
        <form name="loginform" id="formlogin" method="post" action="/home/login">
           <div class="loginContainer">
            <div id="usrFieldsubmit" onclick="document.getElementById('form-
            login').submit()" class="authField input-sm btn-sm">
              <p style="font-size:150%"><strong>></strong>
              </p></div>
               <input type="submit" class="hidden" />
            </div>
           <div class="loginContainer">
            <input id="usrFielduser" name="usrName" class="authField input-sm"
            type="text" placeholder="Käyttäjä" />
            <input id="usrFieldpw" name="usrPW" class="authField input-sm"
            type="password" placeholder="salasana" />
          </div>
        </form>
 }
```

Ylläolevassa esimerkissä luodaan cshtml sivustolle sisäänkirjautumislomake, jos käyttäjällä ei ole autentikaatiopolettia sessiossaan. Autentikaatiopoletti asetetaan käyttäjälle käsittelijässä. Ylläolevassa esimerkissä voimme havainnoida, että lomake laukaisee */home/login* toiminnon käsittelijästä, antaen sille tiedot *post*- metodia hyväksikäyttäen. Käsittelijän puolella tiedot vastaanotetaan ja varmennetaan:

```
[HttpPost]
public ActionResult login(FormCollection logininfo)
  {
        try
        {
          var entities = new TestdbEntities();
          var usrData = entities.user.ToList();
          string rUser = logininfo["usrName"].ToUpper();
          string rPW = logininfo["usrPW"];
          foreach(var usr in usrData)
          {
           if (usr.username.ToUpper().Equals(rUser) && usr.password.Equals(rPW))
              {
                 FormsAuthentication.SetAuthCookie(rUser, true);
              }
          }
          return Redirect(HttpContext.Request.UrlReferrer.AbsoluteUri);
        }
        catch
        {
          return Redirect(HttpContext.Request.UrlReferrer.AbsoluteUri);
        }
 }
```

Ylläolevasta esimerkistä voimme nähdä, että tiedot vastaanotetaan ja osoitetaan muuttujille. Kyseisiä muuttujia tämän jälkeen verrataan tietokannassa oleviin rekisteröityihin käyttäjiin. Jos vastaavat käyttäjätiedot löytyvät tietokannan käyttäjärakenteesta, niin käsittelijä antaa käyttäjän sessiolle autentikaatiopoletin varustaen sen käyttäjän nimellä. Mikäli käyttäjänimi tai salasana eivät täsmää, ohjaa käsittelijä session takaisin pääsivulle.

3.5 **TOIMINNANOHJAUSJÄRJESTELMÄ**

ERP, eli *Enterprise Resource Planning* ohjelmistolla tarkoitetaan sovellusta, joka sisältää toimintoja esimerkiksi tilauksen, laskutuksen, varastonhallinnan ja tuotannon saralta. Toiminnanohjausjärjestelmä pyrkii nopeuttamaan yllämainittuja prosesseja ja automatisoimaan toimenpiteitä. Teollisuuden ja tuotannon alalla keskinäisenä toimintona ERP-järjestelmässä on yrityksen tuotteiden varastosaldojen ja tilausten hallinta. Toiminnanohjausjärjestelmät useimmiten käyttävät hyödykseen myös tietokantaa, johon voidaan tallentaa asianmukaiset tiedot ja lokit myöhempää tarkastelua varten. Tietokanta sisältääkin siis kaiken tiedon yrityksen tuotteista, varastosaldoista, tilauksista, kirjanpidosta, projekteista ja laskutuksista. Tätä tietoa voidaan hyödyntää monella eri tavalla parantaakseen yrityksen toimintaa vieläkin tehokkaammaksi.

3.5.1 **Lemonsoft**

Asiakasyrityksen toiminnanohjausjärjestelmä on nimeltään Lemonsoft. Kyseinen järjestelmä on melko dynaamisesti toteutettu mukautumaan monelle eri toimialalle. Kyseisen toimintaohjausjärjestelmän voi joko tilata palveluna tai lokaalisti asennettuna. Asiakasyrityksen Lemonsoft sijaitsi lokaalisti asennettuna yrityksen lähiverkon palvelimelle. Kuten monet ERP-järjestelmät, myös Lemonsoftissakin hyödynnetään tietokantaa.

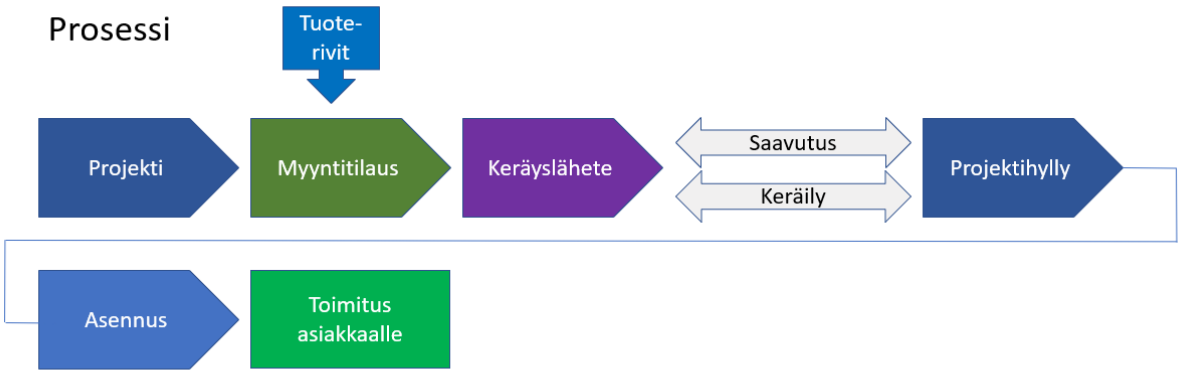
4 **MATERIAALINHALLINTASOVELLUS**

Asiakasyritys halusi kehittää ja parantaa toimintavarmuuttaan ja edistää prosessin kulkua LEAN-ajatusmaailman mukaisesti. Kyseistä kehitysehdotusta aloitettiinkin toteuttamaan materiaalinhallintasovelluksen avulla. Pääasialliset tuotannon prosessit joita pyrittiin parantamaan, olivat seuraavat:

- Tarkempi materiaalinhallinta
- Projektien tuotteiden kokonaiskuvan selkeyttäminen
- Keräyslähetteiden paperisen version sähköistäminen
- Tarkempi tieto projektin kulusta
- Nopeampi puuttuminen tilaamattomiin tuotteisiin.

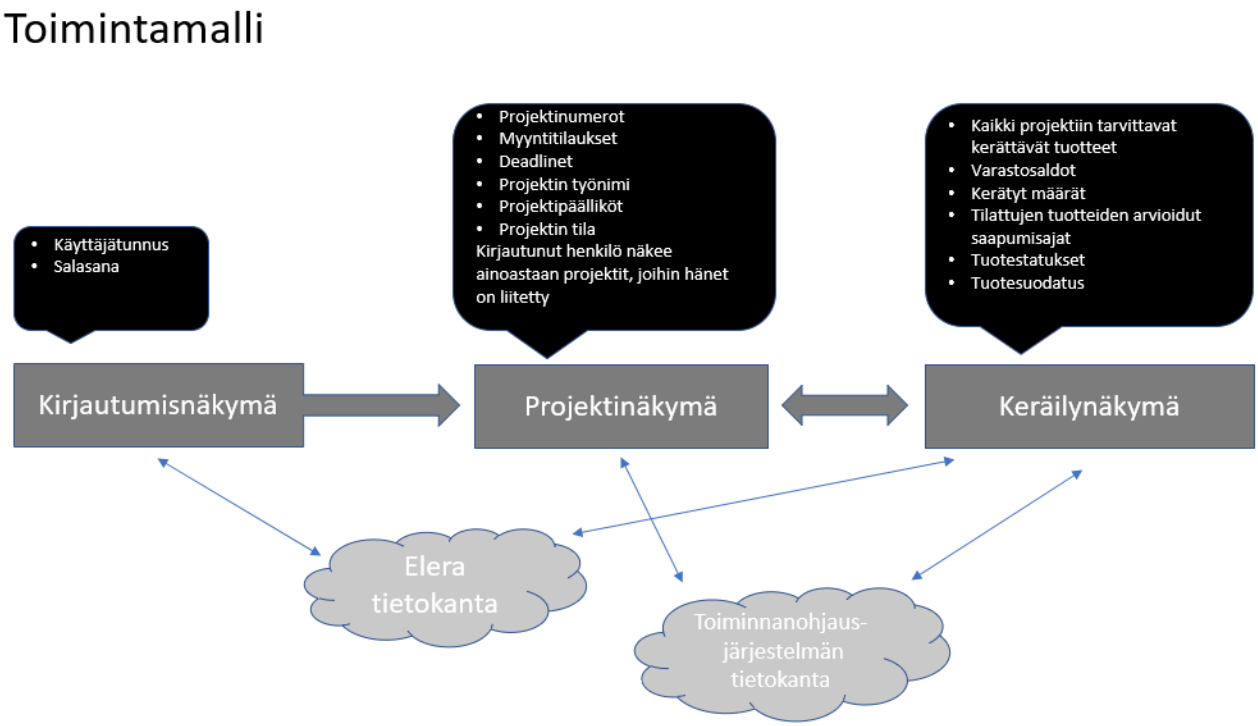
Asiakasyrityksen senhetkinen keräysprosessi käytti hyväkseen paperista keräyslähetettä, jotka saattoivat toisinaan olla jopa kuusikymmentä sivua pitkiä. Tämä aiheutti ongelmia mm. kerättyjen tuotteiden kartoittamisen kanssa. Paperisen keräyslistan kanssa, tuotteita keräävä työntekijä ei myöskään tiennyt, että oliko tuotetta varastossa vai onko tuote tilattu suoraan projektille. Jos tuote oli tilattu projektille, kyseisen tuotteen saapumisajan sai tietää ainoastaan toiminnanohjausjärjestelmästä. Tämä puolestaan aiheutti tarpeetonta kyselyä tilauksista vastaavilta henkilöiltä.

Tuotteita vaativat projektit koostuvat yhdestä, tai useammasta myyntitilauksesta. Jokainen myyntitilaus sisältää tuoterivejä, joita kyseinen projekti vaatii. Alla olevassa kuvassa on kuvattu projektin prosessi.



Kuva 7. Projektin prosessin kulku yksinkertaistettuna

Materiaalinhallintasovellus, jolla prosessia lähdettiin tehostamaan, sai työnimekseen *Sähköinen Keräyslista Elera.* Ohjelman päätarkoituksena on saada nopeammin ja helpommin projekteihin vaadittavat tuotteet esille. Esitettävää dataa ehostetaan myös lisätiedoilla siitä, milloin tietyt projekteihin vaadittavat tuotteet saapuvat. Näin pyritään vähentämään kyselyä vastuuhenkilöiltä ja tehostamaan prosessin etenemisen nopeutta. Pääominaisuutena tässä sähköisessä keräyslistassa on merkitä tuote kerätyksi projektihyllyyn, jolloin projektia asentava työntekijä voi myös tarkastella erinäisten osien asennusvalmiutta.
Sähköisen keräyslistan vaatimuksena oli myös se, että sen täytyy toimia tekemättä muutoksia alkuperäisen toiminnanohjausjärjestelmän tietokantaan.



Kuva 8. Sähköisen keräyslistan toimintamalli

4.1 **Toiminnallisuus**

Sähköinen keräyslista Elera sijaitsee Elkome-yhtiöiden suljetussa lähiverkossa IIS-palvelimella.

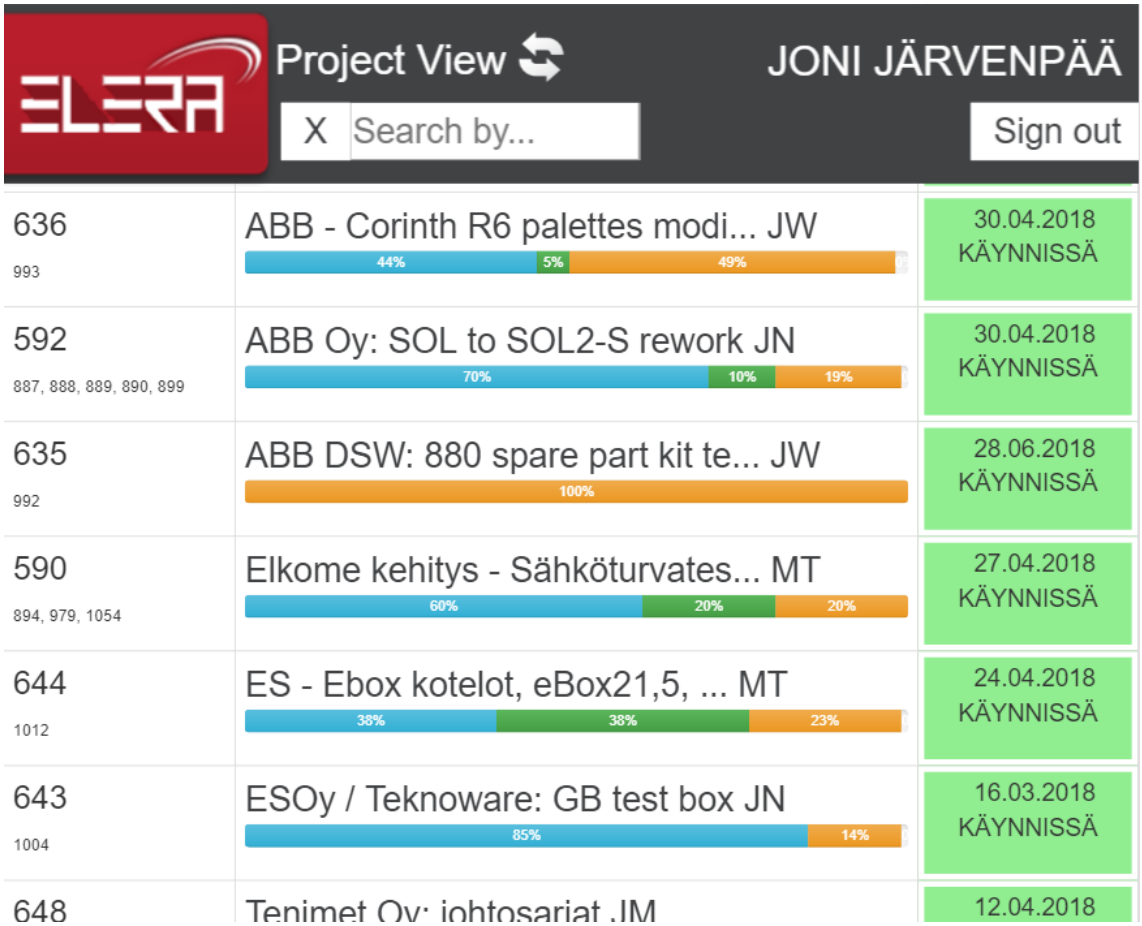
4.1.1 **Kirjautuminen**

Palveluun kirjaudutaan samoilla käyttäjätunnuksilla, mitkä henkilöstöllä on käytössään Lemonsoft toiminnanohjausjärjestelmässä. Käyttäjärakenne helpottaa työntekijälle olennaisten projektien esilletuontia projektinäkymässä.

Kuva 9. Kirjautumisnäkymä

4.1.2 **Projektinäkymä**

Käyttäjän kirjauduttua sisään, näkymä muodostuu henkilölle osoitettujen projektien tai sähköisen keräyslistan roolituksen mukaan. Mikäli käyttäjällä on oletusrooli, henkilö näkee projektinäkymässä ainoastaan ne projekti, jotka ovat avattuja työnjohtajan toimesta ja joihin hänet on osoitettu toiminnanohjausjärjestelmän projektin luontivaiheessa.
Mikäli henkilölle on asetettu työnjohtajan rooli, niin kyseinen käyttäjä näkee jokaisen käynnissä olevan projektin. Työnjohtaja voi avata projektin näkyväksi projektin työntekijöille tästä näkymästä.

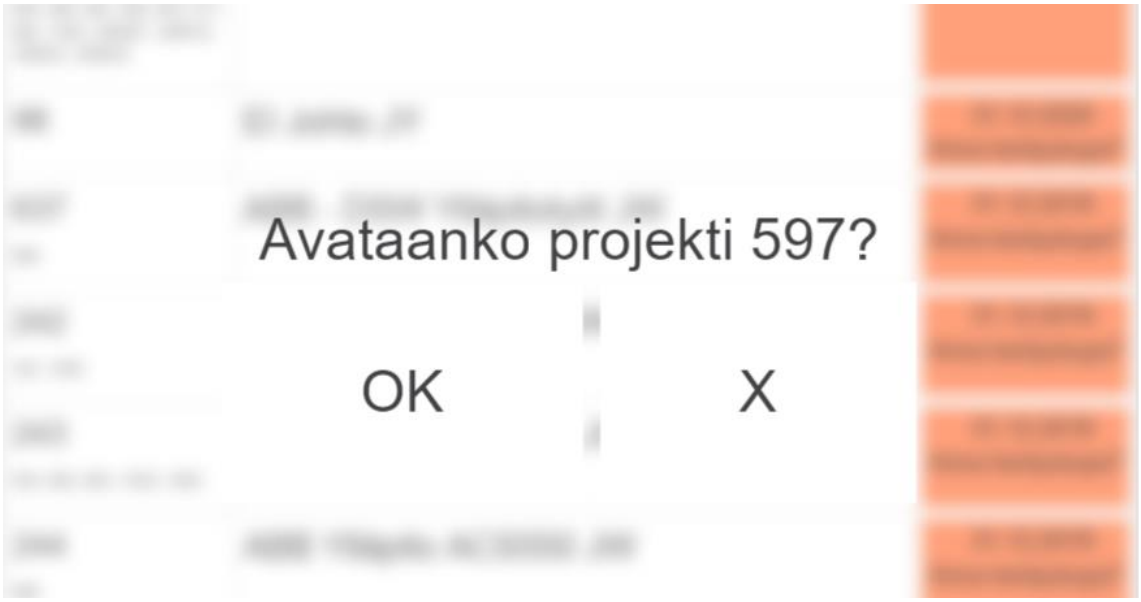


Kuva 10. Projektinäkymä

Projektinäkymästä voidaan myös etsiä projekteja, joko niiden nimellä, tai myyntitilausnumeron/projektinumeron perusteella. Näkymässä vasemmanpuoleisin sarake näyttää projektinumeron ja myyntitilausten numerot projektinumeron alapuolella.
Keskimmäisessä sarakkeessa näkyy projektin asiakas ja nimi, kuin myös projektin työnjohtajan nimikirjaimet. Projektin nimen alapuolella on edistyspalkki, joka kuvaa prosentuaalisesti projektin etenemistä. Värikoodaus on seuraava:

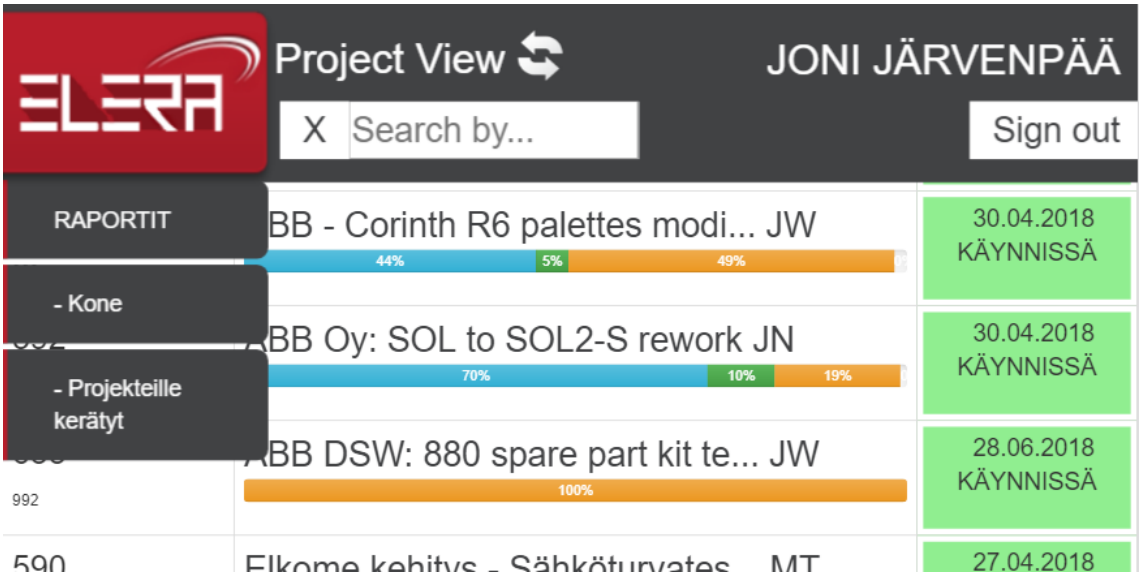
- **Sininen:** Prosentuaalinen osuus tuotteista jotka ovat kerättyjä/saavutettuja projektihyllyyn
- **Vihreä:** Prosentuaalinen osuus, jotka voidaan kerätä päävarastosta kyseisen projektin projektihyllyyn.
- **Keltainen:** Prosentuaalinen osuus tuotteista, jotka ovat tilattuja projektille, mutta eivät ole vielä saapuneet.

Oikeanpuolimmaisessa sarakkeessa näemme kyseisen projektin tilan. Mikäli tilan kohdalla lukee *”Anna keräyslupa?”,* niin voidaan kyseinen projekti avata tarvittaessa projektin työntekijöille ja varastokeräykseen.
Projektin avaus tapahtuu klikkaamalla *”Anna keräyslupa?”* – saraketta, jolloin ohjelma kysyy käyttäjältä varmistusta projektin avaamiseen.



Kuva 11. Projektin avaus

Kyseisestä näkymästä voimme myös saada otteen projektille kerätyistä tuotteista painamalla sovelluksen logoa vasemmasta yläkulmasta. Kyseinen toiminto on ainoastaan saatavilla työnjohto roolituksen omaavilla käyttäjillä. Tämä toiminto tuo esiin valikon, josta saamme otteen Excel muodossa ulos painamalla ”Projekteille kerätyt”- kohtaa listasta.

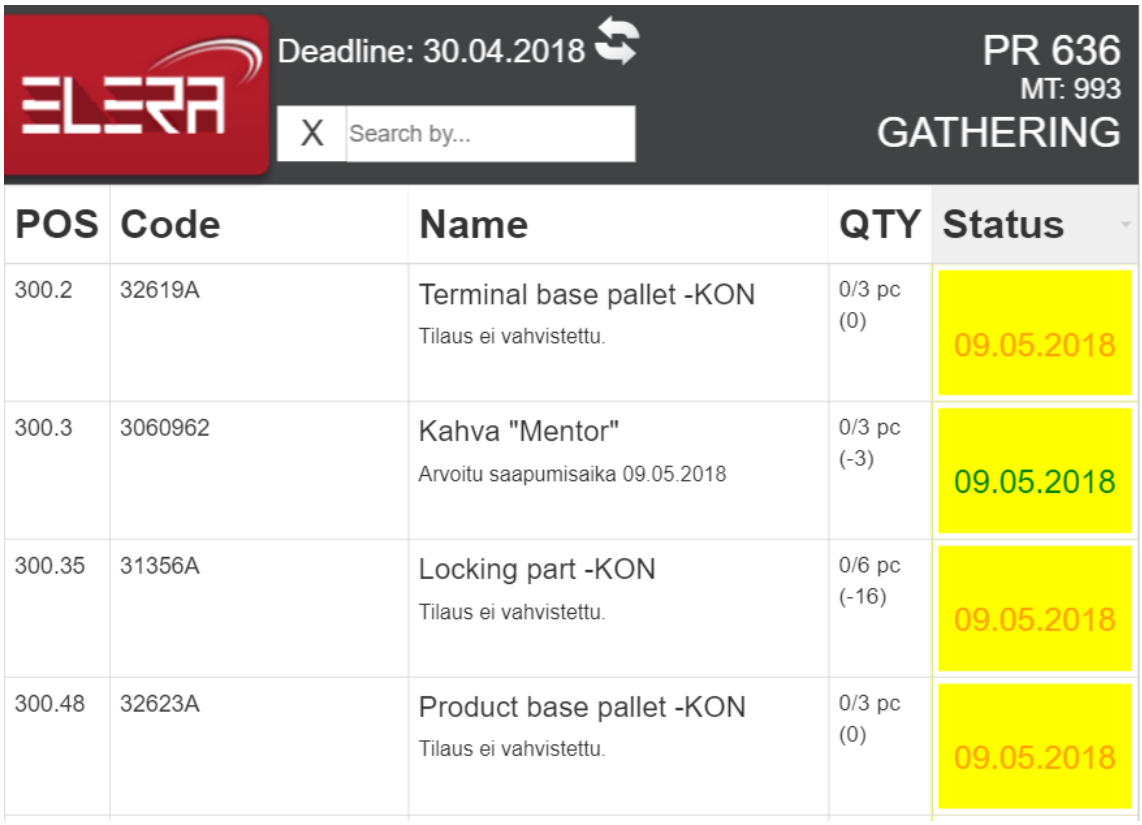


Kuva 12. Raportti osio

Painamalla projektin nimeä, avautuu projektin keräysnäkymä, jossa pääsemme käsiksi kyseisen projektin tuotteisiin.

4.1.3 **Keräysnäkymä**

Keräysnäkymässä näemme vastaavan näkymän:



POS	Code	Name	QTY	Status
300.2	32619A	Terminal base pallet -KON Tilaus ei vahvistettu.	0/3 pc (0)	09.05.2018
300.3	3060962	Kahva "Mentor" Arvoitu saapumisaika 09.05.2018	0/3 pc (-3)	09.05.2018
300.35	31356A	Locking part -KON Tilaus ei vahvistettu.	0/6 pc (-16)	09.05.2018
300.48	32623A	Product base pallet -KON Tilaus ei vahvistettu.	0/3 pc (0)	09.05.2018

Kuva 13. Keräysnäkymä

POS-sarakkeessa näemme tuotteen position, jonka mukaan tuotteet oletusarvoisesti järjestetään keräyslistassa. Yläsaraketta painamalle voimme järjestää tuotteen eri kriteerien mukaan. Code-sarakkeessa näemme tuotekoodin. Name-sarakkeessa näemme tuotteen nimen, ja sen alapuolelle on kirjoitettu pienemmällä fonttikoolla tuotteen tilasta lisätietoa. QTY-sarakkeessa näemme tuotteen saatavuuden. Kyseinen saatavuus on ilmoitettu vastaavalla kaavalla:

tuotteita kerätty / tarvittu määrä Yksikkö
(varastosaldo)

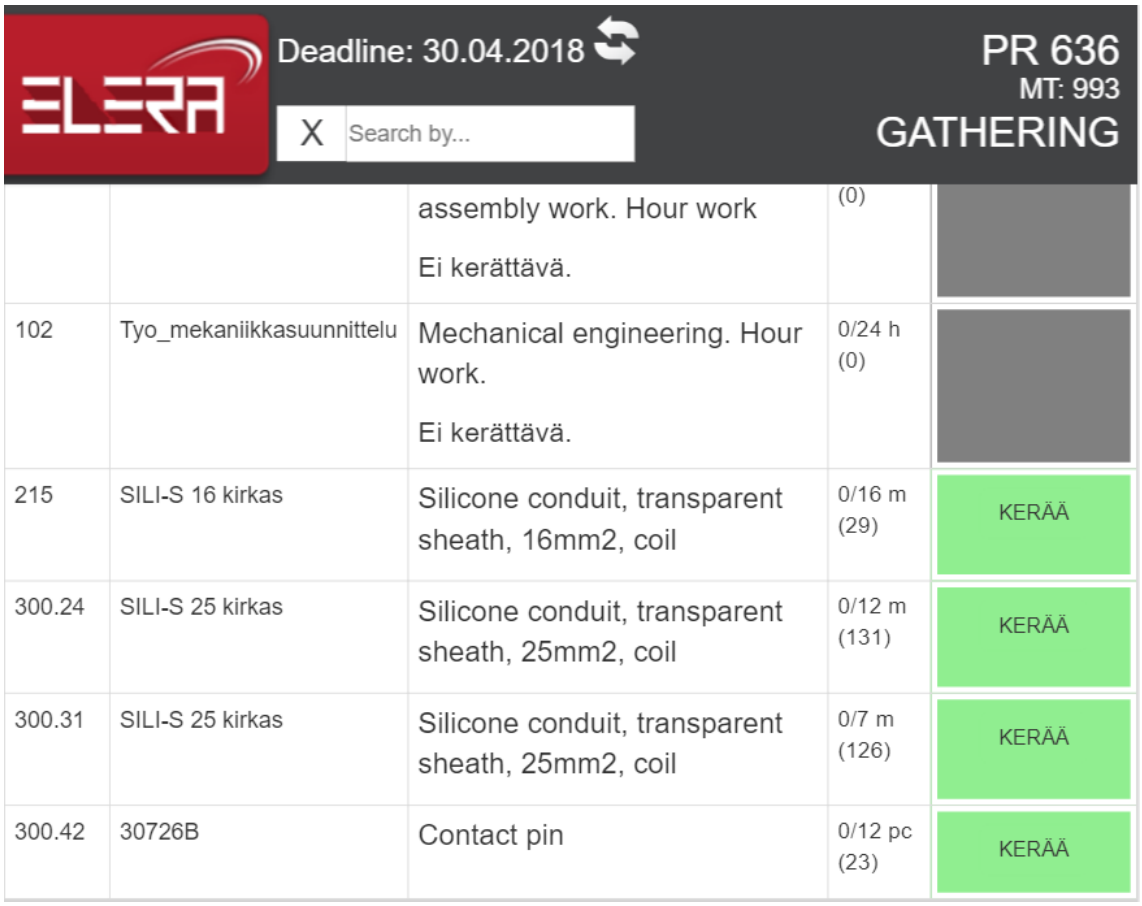
Ylläolevasta kuvasta (kuvasta 11) voimme ensimmäisen rivin kohdalta pääteltä, että tuotteita ei ole kerätty yhtään kyseiselle projektille. Kyseistä tuotetta tarvitaan 3 kappaletta, ja niitä on varastossa 0 kappaletta. *Status* sarakkeessa näemme värikoodatusti tuotteen tämänhetkisen tilanteen. Keltainen status tarkoittaa tuotteen olevan tilattu suoraan projektille. Tällöin laatikon sisällä lukee toimittajan antama arvioitu toimitusaika tuotteelle. Mikäli toimitusaika on oranssilla, tarkoittaa että toimittajan tilausta ei ole vielä vahvistettu. Jos tilausaika lukee vihreällä, niin toimittaja on vahvistanut tuotteen tilauksen. Jos tilausaika on punaisella kirjoitettu, niin toimittaja on vahvistanut toimituksen, mutta tuote on myöhässä.
Status sarakkeen värikoodaus menee seuraavasti:

- **Keltainen:** Tuote on tilattu projektille.

- **Sininen:** Tuote on kerätty tai saavutettu.
- **Vihreä:** Tuote on kerättävissä varastosta.
- **Harmaa:** Ei kerättävä rivi (esim. työ).

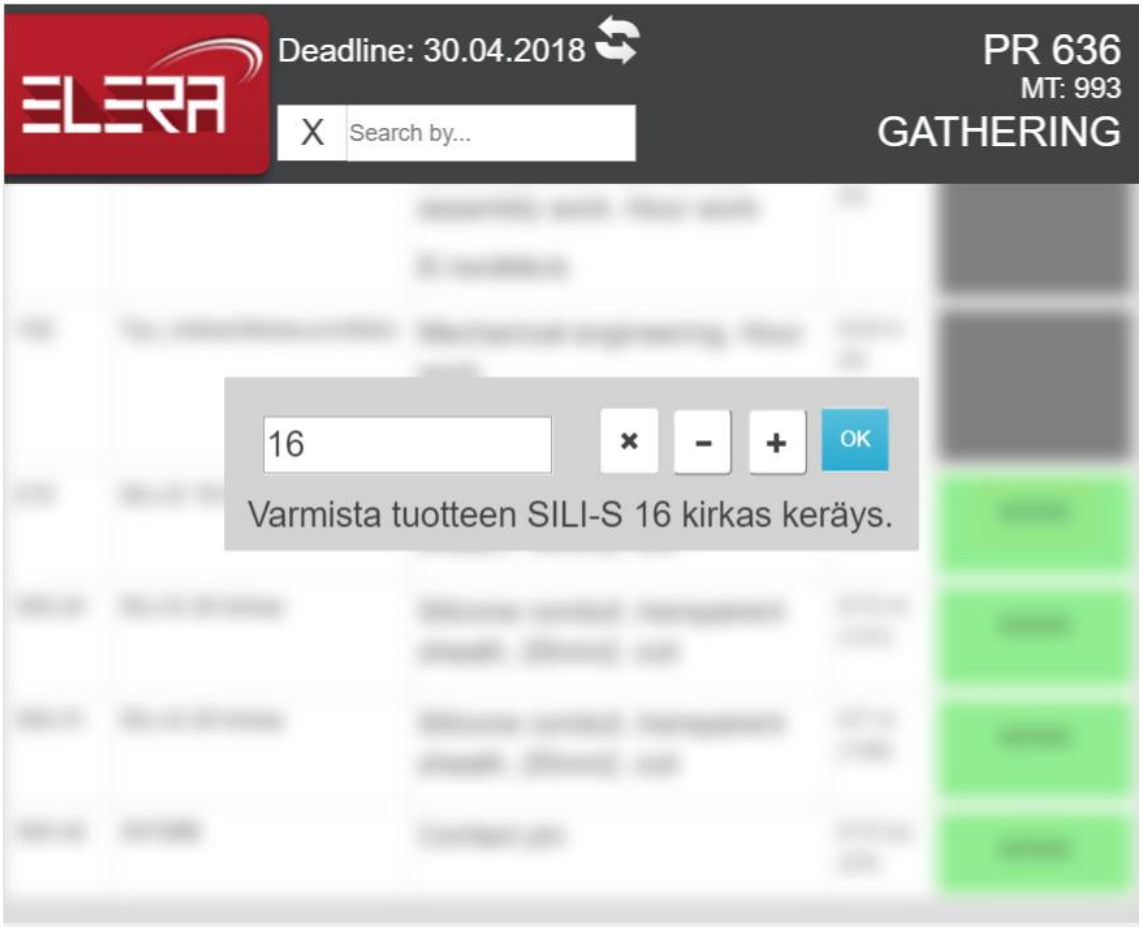
Kyseisessä keräysnäkymässä voimme myös hakea tuotteita Search-kohdasta. Hakutoiminto hakee joko tuoteposition, koodin tai nimen perusteella automaattisesti.

Mikäli tuotteen *status* on vihreänä, niin käyttäjä voi painaa rivillä esiintyvää ”KERÄÄ” painiketta.



Kuva 14. Kerättäviä tuotteita

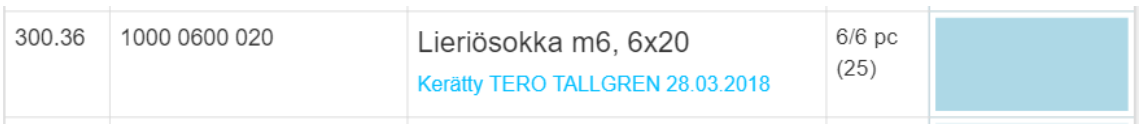
”KERÄÄ”- nappia painamalla näemmä seuraavanlaisen lisäikkunan:



Kuva 15. Tuotteen keräys

Kyseisestä valintaikkunasta varmistetaan tuotteen keräys, jossa voimme vaikuttaa kerättävään tuotemäärään. Mikäli haluamme kerätä ainoastaan osan tuotteista, voimme napeilla lisätä tai vähentää tuotemäärää saldon rajoissa. Tuotteen keräys varmistetaan painamalla sinistä ”OK”-nappia.

Mikäli tuotetta on kerätty yhtä paljon kuin sen tarve projektille on, saa tuote ”kerätty” statuksen, jolloin sen statusväri muuttuu siniseksi. Kerätyt tuotteet näyttävät lisätiedoissa myös sen, että milloin kyseinen tuote on kerätty ja kenen toimesta.



Kuva 16. Kerätty tuoterivi

5 YHTEENVETO

Sähköinen keräyslista Elera on ollut Elkome Installaatiot Oy:n tuotannon käytössä 8.1.2018 lähtien. Kyseistä keräys- ja materiaalinhallinta prosessin parantamista varten työllistettiin varastovastaava, jonka jokapäiväiseen työtoimenkuvaan kuuluu projektien tuotteiden seuranta ja keräily sähköisen keräyslistan avulla. Projektin suunnittelu aloitettiin vuoden 2017 elokuun alussa ja ohjelmiston luominen saman kuun loppupuolella. Suunnitelma projektin toteuttamiseen olisi voinut olla kokonaisvaltaisempi, sillä useita koodimuutoksia on jouduttu tekemään matkan varrella lisäominaisuuksien takia. Käyttöliittymän ulkoasu tehtiin useaan kertaan uudestaan saadakseen mahdollisimman selkeän ja skaalautuvan näkymän käyttäjälle. Palavereita on käyty mahdollisista lisäominaisuuksista sähköiseen keräyslistaan, mutta samanaikaisesti toiminnanohjausjärjestelmä Lemonsoft on uusien päivitysten myötä täyttänyt tarpeet, jotka oltaisiin muutoin integroitu Sähköiseen keräyslistaan.

Kokonaisvaltaisesti sähköisen keräyslistan tavoitteellinen prosessin parantaminen onnistui hyvin, sillä projektiin kohdistuvat tuotekysymykset niin saatavuuden ja saapumisajan ohjautuvat varastovastaavalle, joka näkee tiedon Sähköisestä Keräyslistasta. Tämä vähentää ajankulutusta toimistopuolen vastuuhenkilöiden konsultaatiosta joka tukee LEAN-ajatusmaailmaa. Sähköiselle keräyslistalle tehdään päivityksiä noin kerran kuukaudessa riippuen siitä, minkälaisia parannusehdotuksia käyttäjät keräävät ohjelmistosta.

Parannusehdotuksena jatkokehityksen kannalta tietokanta hakuja voisi optimoida selkeämpään muotoon, sillä osa LINQ-hauista ovat venyneet yli kymmenrivisiksi.
Kehittämistyö Sähköisen keräyslistan parissa on koko prosessin ajan ollut sujuvaa ja opettavaista. Kyseinen työ oli ensimmäinen iso projekti jossa sain mahdollisuuden hyödyntää opiskelemiani asioita käytännössä ja lisämahdollisuuden oppia paljon uusia tekniikoita.

LÄHTEET

Devlab (2018). ERP:in hyödyt pähkinänkuoressa. Haettu 22.4.2018 osoitteesta https://www.devlab.fi/erpin-hyodyt-pahkinankuoressa/

Hejlsberg. A. (2007). LINQ: NET Language-Integrated Query. Haettu 22.4.2018 osoitteesta https://msdn.microsoft.com/en-us/library/bb308959.aspx

Lemonsoft (2018). Toiminnanohjausjärjestelmä. Haettu 22.4.2018 osoitteesta https://www.lemonsoft.fi/ratkaisu/toiminnanohjausjarjestelma/

Microsoft (2018a). Overview of ASP.NET Core MVC. Haettu 20.4.2018 osoitteesta https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-2.1

Microsoft (2018b). Authentication scenarios for Azure AD. Haettu 22.4.2018 osoitteesta https://docs.microsoft.com/en-us/azure/active-directory/develop/active-directory-authentication-scenarios

Microsoft (2018c). ASP.NET Forms Authentication Overview. Haettu 22.4.2018 osoitteesta https://msdn.microsoft.com/en-us/library/7t6b43z4.aspx

W3resource (2018). Simple Database. Haettu 22.4.2018 osoitteesta https://www.w3resource.com/sql-exercises/movie-database-exercise/basic-exercises-on-movie-database.php

Quality Knowhow Karjalainen Oy (2018). Lean ja Johtaminen. Haettu 20.4.2018 osoitteesta http://www.sixsigma.fi/fi/lean/yleinen/lean-ja-johtaminen/